9. Object-oriented testing

Nowadays most of the software is designed and implemented using object-oriented techniques. How does testing of object-oriented systems differ from traditional procedural programs?

- Based on [Pezzè&Young 07]
- Essentially the techniques for testing procedural programs work also with testing object-oriented programs
- Because of the specific properties of objects some crucial parts might remain untested
- For this reason, a number of object-specific techniques have been developed, to help test the areas that are otherwise left untested
- Affects testing mainly at unit and integration levels

- The pros of object-oriented systems from tester's point of view
  - Object-oriented programming has brought more attention on the design of the programs (design patterns etc.)
    - On the other hand, the same problem can be solved in very different ways
  - Methods are usually shorter than the corresponding procedures
    - Usually the ”shortness” of a method means easier unit testing
    - On the other hand, there might be more methods
  - Test cases are potentially reusable just like the code

- The greatest difference with the testing object-oriented programs is that an object has a state
  - The behavior of the system in certain moment depends on the inner states of its objects
  - The techniques that do not take into a consideration the states of the objects are not necessarily enough for testing an object-oriented system
  - The lifecycle of an object from construction to destruction

- Another important distinction is data hiding
  - Some of the attributes of an object that define its state are hidden from external viewers
  - Information hiding can improve the quality of the code and reduce the need for testing
  - One must remember to check the hidden values of the variables when the behavior of the system is inspected

- Inheritance
  - Can the child class be left untested on the basis that its parent class has already been tested?
  - The testers have to decide
    - Is it necessary to develop new test cases for the methods of the child class
    - Is it necessary to run test cases of the base class again for the child class
    - Which of the methods in the child class need not be tested

- Polymorphism and dynamic binding
  - The class of an object is not determined at the time of the compilation, so the real target of a method call is decided at run time according to the state of system
    - This makes it more difficult to design and understand the code
  - Traditional techniques that assume static binding do not cover all the possibilities of binding (behavior)
  - Different binding possibilities must be tested
    - Can we apply decision coverage?
  - In the case of multiple calls also their possible binding combinations should be tested

- Abstract classes
  - Abstract classes may be important parts of component interfaces or libraries, but they cannot be instantiated or tested
  - Sometimes it might be impossible to know how these classes in question are to be used in practice
  - Testers must try to make educated guesses about the important ways to use abstract classes and test accordingly

- Exceptions
  - Exception mechanisms of object-oriented languages are used frequently
  - An exception can be thrown in a completely different part of the code from where it is processed
  - Dynamic binding brings in complexity of its own
  - One should pay especial attention to testing exceptions

- Concurrency
  - The usage of threads is encouraged or even required in object-oriented languages
    - For example user interface libraries of Java
  - Unfortunately concurrency is a major issue when it comes to testing
    - Usually an event A happens before event B, but is this always true or does it depend on the scheduler?
    - Does mutual exclusion work correctly in all situations?

- Objects affect testing on the lowest levels of the V-model
  - Mostly in unit and integration testing
  - In other phases the traditional methods work well
- As a rule of thumb it can be said that each special property listed above should be tested separately
  - An exception is testing properties that interfere with each other, such as inheritance and behavior that depends on the state

- Unit testing
  - One should not think methods as units to be tested, as their behavior depends often on the state of the surrounding object
  - Class appears to be the most suitable unit for testing
    - One should remember prioritizing: not all classes are equally important
    - Classes cannot be tested without testing its methods

- Unit testing of a class
  - If a class is abstract, objects for testing are instantiated from its child classes
    - If there are no child classes, they must defined only for testing purposes
  - Design test cases that test the calling of the inherited and overridden methods
    - Analyze which of the inherited methods have to be tested again
    - Analyze whether the test cases of the base class can be used again
    - Don't forget constructors and destructors (releasing resources)

  - Gray box testing: if the behavior of a class is specified with a state machine, design a suite of test cases based on the state machine
    - If there is no state machine specification, it might be worthwhile to make one to ease the testing
  - White box testing: Add new test cases with the help of the code and its structure
  - Design test cases that test exception handling
    - Exceptions that are thrown, caught or handled by the methods of this class

- Extend the test suite with tests that test the polymorphic method calls that target the methods of this class
- It might be worthwhile to pay attention to the interaction within the class
  - Methods can use the same attributes
  - Methods can call other methods of the same class

- Integration testing of classes
  - Compare to incremental integration strategy
  - Start with small clusters of classes whose size is increased gradually
  - First one must find out the dependencies between the classes
    - Class A depends on class B if
      - The instances of A call the methods of the instances of B
      - The instances of A contain references to the instances of B
    - The inheritance relations and abstract classes can be ignored
    - Note! In case of call-backs (for example user interface libraries) the application always depends on the library and not vice versa

– In well-designed software the dependencies do not generally form loops, meaning that classes of the clusters and dependencies between them form a tree-like structure
  - If there are loops, they must be disassembled by replacing a class with a stub
– Testing is to be started from the leaf classes, which do not depend on other classes
– Include base classes of the leaf classes, and so on, progressing toward the root of the tree

– In every phase black box tests that test functionality are designed/executed for the cluster
– Additionally, if necessary, white box tests are designed/executed
  - For example until the required coverage is reached
– Test cases must especially test also
  - Exceptions that are thrown and handled in different classes (within the cluster)
  - The polymorphic method calls between the classes of the cluster

- The scaling of object-oriented testing based on risks according to Pöyhönen and Stenberg (NRC) (www.cs.tut.fi/tapahtumat/olio2002/):
  - Small risk:
    - 100% method coverage
      - The usage of equivalence classes (legal equivalence classes)
      - The usage of boundary value analysis (boundary values of legal equivalence classes)
    - Testing of essential inheritance relationships
    - Dynamic binding is tested with inheritance
    - While testing memory management the most important constructors and destructors are tested
    - Some parts of the code are inspected
    - Static analysis of the code
    - Object models are inspected informally

  - With moderate risk, in addition of the previous:
    - 100% method coverage
      - The usage of equivalence classes (legal + illegal equivalence classes)
    - More inheritance relations
    - Testing of exceptions
    - Dynamic binding is tested separately
      - Although not all combinations
    - All constructors are tested
    - Object models are inspected

- In the case of great risks, all the previous plus the following:
  - 100% method coverage
    - The usage of boundary analysis (boundary values of legal + illegal equivalence classes)
  - All inheritance relations are tested
  - All combinations of dynamic binding are tested
  - Systematic code inspections

Mock objects

- Real problems detected in the unit testing of objects:
  - The replacement of dependencies with stubs can be lots of work (databases, protocol stacks, user interfaces)
    - The size of the unit under test grows
  - The repeatability of tests is often weak
    - Ad-hoc type of manual labor
  - The system has not been designed to be tested
- Mackinnon, Freeman, Craig: ”Endo-testing: Unit testing with Mock Objects”, XP2000
- Can be seen as intelligent stubs

- A more structured way of creating and managing the test code that replaces the original services
- Requires the application of good o-o design conventions from the target
  - The usage of mock objects may require refactoring
- The benefits of Mock objects
  - Setting of desired state is easy
  - Changing the functionalities is easy
  - Checking the state is easy

- An example: testing of a simple currency converter:

```
public class Converter_math {
   public static void Eur2fim(String euros_str, PrintStream out){
      double marks_d = 0.0;

      try {
         marks_d = Double.parseDouble(euros_str)*5.94573;
         out.println("Result: " + euros_str + " euros correspond to "
                     + marks_d + " marks.");
      } catch (NumberFormatException e) {
         out.println("Error in the amount of euros: " + euros_str);
      }
   }
}
```

– Main program:

```
public class Converter_main {
   public static void main(String[] args) {
      if(args.length >= 1) {
         if(args[0].equals("testing")) {
            test_converter_math();
         } else {
            Converter_math.Eur2fim(args[0], System.out);
         }
      } else {
         System.err.println(“Usage: Give the amount of “ +
         “euros to be converted as a parameter");
      }
   }
}
```


– Testing using mock objects:

```
public static void test_converter_math() {
    MockPrintStream mock = new MockPrintStream(System.out);
    // Test case 1
    mock.setExpectedPrintlnCalls(1);
    mock.setExpectedStringSegment(“Error in the amount of euros: "
                        + “Error state");
    Converter_math.Eur2fim(“Error state", mock);
    mock.verify();
    // Test case 2
    mock.setExpectedPrintlnCalls(1);
    mock.setExpectedStringSegment(“Result: 100 euros correspond to ” +
                        “594.573 " + "marks.");
    Converter_math.Eur2fim("100", mock);
    mock.verify();
 }
```

- Application frameworks/libraries that offer different methods and environments to use mock-objects
  - POCMock/NMock (.NET)
  - Mockrunner (J2EE)
  - EasyMock
  - DynaMock
  - …

- The developer is responsible for building the testability
  - Design patterns help
  - The responsibility cannot be transferred to the tools or methods
- Unit testing requires hard work and an attitude change, especially in the beginning when the test environment has to be set up
- Using Mock objects may result in better design of the unit under test

Testing patterns

- The last few years have brought about testing patterns that can be used in testing in a way similar to design patterns in object-oriented design
- As an example self-shunt unit testing pattern
  - The idea is that the object modeling the test case relays itself as a parameter to the system under test pretending to be a ”real” object with which the test target should communicate with

10. Testing of information security

Couple of years ago information security was of interest to only a small group of people. Unfortunately nowadays everyone must be interested in it. So how should things related to it be tested?

- Adapted from [Whittaker&Thompson 03]
- Testing of information security is important and its importance will only grow in the future
  - Computers (PC, mobile phone, PDA etc.) are commonly used to store confidential and secret information
  - Network is more and more often used to transport this kind of information from one place to a another
  - If you don't test the information security, the criminals will do it for you!

- As discussed before, some errors hiding in the code can cause disturbance for example in the functionality visible to the user, some others can cause performance issues
- In contrast, in the testing of information security the point is to find errors that in some situations endanger the security of the information in the system
  - For example the passwords of an online bank are stored in the cache of the web browser even if it is cleared at the end of the session

- Software can work otherwise entirely correctly, but may still contain errors that endanger information security
- The requirements related to information security are not yet widely understood
  - The network is filled with software made with no regard on information security
  - For example operating systems have to be updated constantly with security patches

- Needs for security testing can be analyzed based on risk analysis, which helps us to identify the most important issues in protecting our data and business
- Security can and should be ensured on many levels, such as the physical placement of the server vs. fire walls vs. user authentication
- In Finland, public administration has provided instructions for users and developers of information systems concerning security issues:

  http://www.vm.fi/vm/fi/16_ict_toiminta/009_Tietoturvallisuus/02_tietoturvaohjeet_ja_maaraykset/index.jsp

- OWASP Top 10 – 2010 (www.owasp.org)
  - A1 – Injection
  - A2 – Cross-Site Scripting (XSS)
  - A3 – Broken Authentication and Session Management
  - A4 – Insecure Direct Object References
  - A5 – Cross-Site Request Forgery (CSRF)
  - A6 – Security Misconfiguration (NEW)
  - A7 – Insecure Cryptographic Storage
  - A8 – Failure to Restrict URL Access
  - A9 – Insufficient Transport Layer Protection
  - A10 – Unvalidated Redirects and Forwards (NEW)

- Information security is about keeping secrets
- Typical concerns related to information security are for example:
  - I want to sell my software for a good price and make sure that no one can make illegal copies of it
  - I want only specific people to get their hands on certain parts of the software (access control)
  - I want to protect my server software (and still allow the execution of code uploaded by the user)
  - I want to keep the information used by my software secret on the disk, in memory and when transmitted over a network

- Unlike in traditional testing, specifications are less useful in testing information security
- Instead of testing whether the software fulfills its specification, one should examine more closely
  - What side effects the implementation allows
    - As a result of buffer overflow foreign code might be executed
  - How does the software interact with its environment
    - Operating system calls, network traffic etc.

- The threats that come through the user interface
  - The protection of the user interface from unauthorized users for example with a password
    - Does the user identification work properly?
    - Are weak passwords accepted?
      - Is e.g. username accepted as a password?
    - If a user may access only some of the information (for example a home directory in a multiuser operating system) do the restrictions work correctly?

- Can one obtain confidential information for example by using the copy/paste or screen capture features?
- Is it possible to input illegal inputs such as too long strings?
  - A result can be for example a buffer overflow, which can lead to execution of foreign code
  - The system may also crash, which can mean a denial of service, that is the services offered by the system become unavailable

- Threats coming through the file system
  - The connections of the software to the file system are often poorly tested
  - Yet secrets such as passwords, license keys, etc. are saved in files
  - In the Windows world, the registry is an especially bad place to save secrets
    - In the Unix world the same naturally applies to configuration files of programs (.emacs, etc.)
  - Fuzz testing based on file formats can used to test the robustness of the implemented file operations

- Threats coming through the operating system
  - If the program contains within its memory encrypted secret information, this is often safe; problems appear when information is handled unencrypted
  - If the resources of the operating system run low, such as the amount of available memory, the result may be for example
    - Denial of service
    - Uncontrolled crashing of the program
      - Can secret information be dumped to a hard drive in an unencrypted form?

- The threats coming through other software
  - Programs make more and more use of other programs through different interfaces
  - What happens if another program becomes a victim of an attack or crashes?
  - The tester has to find out the connections to other programs and identify situations where these might cause threats
  - For example, when communicating through a network one should make sure the program functions correctly with illegal packets, too short and too long packet frames etc.
    - Fuzz testing is also needed here
    - Similar worries are related to regular interfaces

- Protecting the software itself
  - Sometimes the program contains algorithms, etc. that give an competitive edge
  - In such a case it might be sensible to protect against reverse engineering trying to figure out the contents of the code using e.g. a disassembler

- Often the attack may come from many fronts at the same time
  - For example blocking access to some library and sending an illegal input through the user interface at the same time
- Sometimes built-in ”features” of the program might offer holes in security
  - Instrumented test code may have been forgotten in the binary code, offering ready hooks for running test cases
- A developer of a single component might have been ignorant of the whole the component belongs to
  - If for example the data relayed as a parameter is secret, it must not be written to disk unencrypted even temporarily

Example attacks

- Block the calls of a application to the (dynamic) libraries
  - A tool is used to observe which libraries the application calls in which situation and the calls to a certain library are blocked
  - The purpose is to test that an application does not endanger information security even if the library is not available

  - Often application developers assume that libraries are always available
    - This is an exceptionally dangerous assumption if a library function is used to ensure something related to information security (for example user privileges)
    - If a call is made to a blocked library function, the result is generally the start of exception handling
      - Often exception handlers are not tested as carefully as other code

- The result may be a crash of the application or the dumping of secret information to the screen or disk
- The application may also seem to continue to function normally, even though the service provided by the library function is not available
  - This can be a sign of the program leaving a return value unchecked, for example
  - The result may even be the granting of incorrect privileges to the user

- Look for suspicious options and their combinations
  - Testing the options that can be given through both the command line and the graphical user interface is difficult because of the large number of combinations
  - If one concentrates only on the most essential options for the user, it is possible that as a result the applications runs partly untested code
  - One should look for combinations of options whose synergy may compromise information security
  - If the software in question is a new version of an old program, one should examine the previous version and its documentation

- If an option has disappeared from the documentation of the new version, test if the implementation still supports it and if it does, does it work safely

– User input might only be verified for some of the options
  - If an application asks for an address, and country can be selected from preselected set of countries using a menu, the checking of a postal code might depend on which country is selected
    – A program might accept an overly long postal number that might contain control symbols…
      » Would it be possible to input SQL-queries and execute them?
    – Or is the postal code checking algorithm implemented also for the addresses of Ivory Coast?

- Port scanning
  – Applications that use network must protect themselves against port scanning
  – A port scanner is used as a help for testing
  – The usage of application specific ports should especially be tested (numbers 1024-65535)
  – If an application is trying to hide an open port by returning an error message, the message must be of the standard format so as not to raise suspicion

- Find alternative ways to do the same thing
  - How many different ways are there to open a Word document in Windows?
  - Have all the ways been tested and shown to be safe?
  - The authors of the book ”How to break software security” found a security hole in Windows XP in the following way:

- Because Windows Explorer can be used for many same purposes as Internet Explorer, let's read emails with IE using MS Exchange Server’s Outlook Web Access interface so that first IE is started through WE.
- After the session we close the WE and IE windows
- Next we contact the email server with WE and what do you know, we get in without a password query!
- When we try to read individual messages with a separate window the password is asked, but when using the preview of Outlook, the password is not asked
- The lesson: Information security was breached when an optional way to do the same thing was used
  - Replaced Internet Explorer with Windows Explorer
  - Replaced an Outlook window with a preview
- Note! This bug has since been fixed

- Problems of testing of information security
  - Testing has to find all the security holes in the software, the intruder needs only one
  - The errors related to information security are usually much less noticeable than for example functionality or performance issues

11. Static testing

The tool chest of a tester and a quality assurance professional contains techniques that complement each other. In the following we present some group work techniques, as well as discuss automatic code analysis.

11.1 Inspection

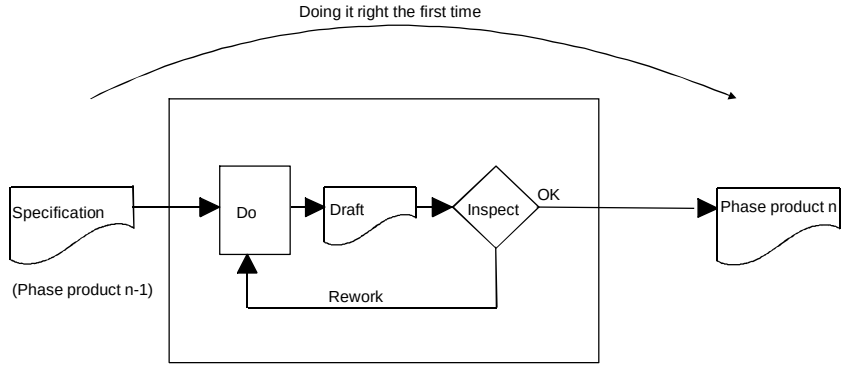
- Composed from the source: Ahtee, Haikala, Märijärvi: Inspection (training material, in Finnish)
- Inspection – IEEE 610.12-1990 (Standard Glossary of Software Engineering Terminology):
  - A static analysis technique that relies on visual examination of development products to detect errors, violations of development standards, and other problems
  - Types of inspections include code inspection, design inspection
- Internal inside a project, 3-6 participants
- Very formal compared to other group working techniques
- Flexible scheduling, multiple inspections during the project
- Only small artifacts are inspected at a time

- Fagan developed the technique at IBM in the 1970's
- Inspections are considered the most effective technique for quality assurance
- Inspections support the process and project in the following ways
  - By trying to remove the errors at the earliest possible stage
  - By making the progress in the project visible
    - An accepted inspection corresponds to reaching of a milestone
  - By sharing information to multiple persons

- An inspection takes at least two hours, during which at most
  - 50 pages of a document or
  - 500 lines of code

  can be inspected
- 50-80% of errors are found
- Inspection consume about 5-15 % of the work time
- Very cost effective!
- Although the method does not add very much to the workload, commitment from the management is needed

- Inspections in the software process



- Inspection can be applied to following artifacts, for instance
  - An offer, an agreement
  - Functional specification
  - Project plan
  - Technical specification
  - Test plan
  - Code
  - Document that is be delivered to the user (manual)
  - Study material

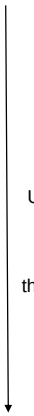
- Roles in a inspection meeting
  - A chairman
  - The author of the phase product
  - Secretary = often the author
  - Inspector = everyone
  - Reader
    - Can be the author
    - Only in the code inspections
  - An expert of the area of the application, for example a specialist, user experience specialist, proof-reader, testing expert, etc.
    - Someone can focus on checking the memory management, somebody else can check loops, algorithms, interfaces, etc.

- Rules of thumb
  - If everything cannot be inspected, only the most important parts are inspected
  - Careful preparation is important
    - The material to be inspected has to be given out to the inspectors 3 days before the inspections
    - The inspection meeting should be canceled, if there are no prerequisites for its success
      - For example phase product is too incomplete
  - The product is always under evaluation, not its author
  - The goal is to find new problems – not solving them
  - The fixes for cosmetic errors can be delivered to the secretary after the inspection meeting

- One should consider the rules that facilitate the inspection
  - For example “does the requirement really consider the real need of the customer”
- The rules should be considered carefully according to the artifact, organization, process, risks, etc.
  - With just a few rules one can reach good results (Marko Komssi, EuroSTAR 2004)
  - The accuracy of the rules can be increased as the process matures
- At IBM they have develop a new version of inspections that does require preparation:

E. Farchi, S. Ur: “Selective Homeworkless Reviews”, Proceedings of the 2008 International Conference on Software Testing, Verification, and Validation IEEE Computer Society Washington, DC, USA

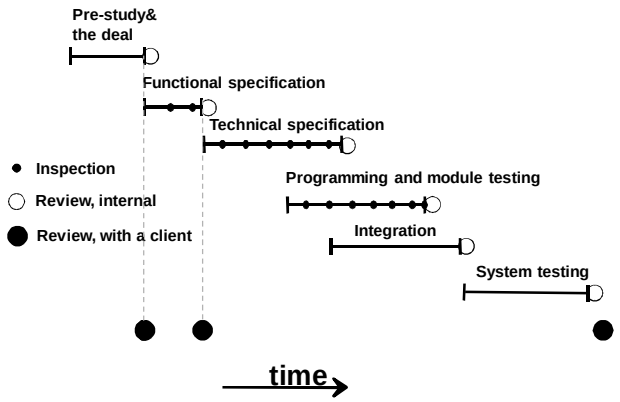
- The maturity levels of inspections
  - Inspections are held
  - Some benefit from inspections
  - Inspections are effective
  - Statistics are collected from inspections
  - Bug analyses and checklists are produced according to the experiences
  - The collected information is used for improving the inspection and the software processes



11.2 Review

- Review or a technical review
- IEEE 610.12-1990:
  - A process or meeting during which a work product, or set of work products, is presented to project personnel, managers, users, customers, or other interested parties for comment or approval. Types include code review, design review, formal qualification review, requirements review, test readiness review.
- Is used for ending a phase
  - For example functional or technical specification phase
- A formal check to see if all the ending requirements of the phase have been met
- The goal is to make the advancement of the project visible
  - The goal is consensus, not so much bug finding

- Reviews and inspections of the project [Haikala&Märijärvi 06]:



- Note! The terminology is not consistent: in some cases inspections are called reviews, or vice versa
- Because these are fundamentally different techniques, it is worthwhile to find out beforehand whether the goal is to inspect or review

11.3 Walkthrough

- IEEE 610.12-1990:
  - A static analysis technique in which a designer or programmer leads members of the development team and other interested parties through a segment of documentation or code, and the participants ask questions and make comments about possible errors, violation of development standards, and other problems.
- Code walkthrough with the purpose of finding errors
- The developer explains what he thinks the program does

- Walkthrough compared to the inspection
  - Emphasizes the role of the developer in the event
  - Is less formal
  - Requires less training
  - Finds fewer errors
  - Is usually less cost effective

11.4 Static code analysis

- Idea: To analyze the source code automatically without executing it
- The purpose is
  - To find errors from the code
  - To notice deviations from the normal coding conventions (style guides)
  - To generate documentation of the code
  - To calculate values for metrics of the program such as the length, complexity

- The techniques to be used are based mostly on information and control flow analysis, constraint solving, etc.
- The first static code analysis tool introduced for the greater audience was Lint that came with UNIX
- The purpose of Lint was to find typical errors from C code that the compilers of that time did not detect

- What kind of errors can be found?
- For example
  - Syntax errors
  - Same code in multiple places, dead code (never executed)
  - Maintenance and portability problems
  - Uninitialized variables
  - Unused return values
  - Misuse of pointers
  - Buffer overflows, other information security issues

- Well-known static analysis tools:
  - PC-Lint, Coverity, PolySpace, KlocWork
  - Usually costly (except the first one)
  - Scalability can become an issue
  - The number of false alarms might also become an issue
  - Can prove valuable very quickly by finding errors in critical code

- Where to find help for documentation?
- For example:
  - Javadoc style API-documentation
  - Doxygen style graphical model (UML) of the software
    - www.doxygen.org
  - Call hierarchy